Volume 2, No. 1, Jan-Feb 2011
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

Development of Distributed Monitoring Framework

Harikesh Singh*, Dr. Shishir Kumar
Department of Computer Science & Engineering
Jaypee University of Engineering & Technology
Guna, India
harikeshsingh@yahoo.co.in
shishir_ks@yahoo.com

Abstract: The problem of network intrusion has been tried to solve by many different approaches. Packet sniffing is one of the most simplistic approaches. In this rapidity growing technological world where system safety questioned by different network intrusions is growing per minute, network monitors help solve these problems of unwanted having and problems arising from the periodical failure in the working of wiring and protocols due to their complication. Network monitor can be connected to the network to analyze what is happening and attempt to identify the cause of the problem. "Monitoring data on internet" is used to develop a log file of packet contents that are transferred between two different systems connected via a local area network. Their average data transfer rate and other details are analyzed by the software. Installing our software the system is aimed to act like an approximate network monitor in some contexts as it helps in evaluation of data flow from system. It can be successfully used for capturing data transfers between your PCs and any other network resource.

Keywords: distributed system, distributed monitoring, network, data transfer, performance monitoring

I. INTRODUCTION

In the distributed monitoring framework, the distributed systems engross the compilation, analysis, and demonstrate of information regarding the interactions among parallel executing processes. This information and its display can sustain the debugging, testing, performance assessment, and dynamic documentation of distributed systems. When building distributed systems, it has been pragmatic unexpectedly low performance [1]. The bottlenecks can be in any of the following components: Applications, Operating systems, Disks or network adapters on either the sending or receiving host, Network switches and routers, and so on.

The distributed monitoring of components is significant for enabling high-performance distributed computing. Monitoring data is needed to determine the source of performance problems and to tune the system and application. Fault detection and recovery mechanisms need monitoring data to determine whether a server is down and to decide whether to restart the server or to redirect service requests elsewhere. A performance prediction service takes monitoring data as input to a prediction model, which is in turn used by a scheduler to determine which resources to assign to a job. Analysis tools will be able to use the monitoring data for real-time analysis, anomaly identification, and response [2].

II. RELATED WORK

A. *Tools Used In Traffic Monitoring*

This task can be time consuming and cumbersome and in most cases accurate information about the network is not obtained. Some researchers have developed modular software architectures for extensible system, however only a few of these systems are optimized to handle large amount of data and continuous monitoring. The libpcap tool has greatly simplified the task of acquiring network packets for measurement. The limitation of the tool is its inability to analyze the captured data, it will only capture the data and the programmer or network administrator is left to carry out analysis manually. Other researchers have developed systems for streaming data through protocol layers and routing functions, but not much attention has been given to the analysis of large/huge or broad data collected over time[3][4].

The most common use for these tools is graphing the In Octet and Out Octet counters on router interfaces, which respectively provide counts of the number of bytes passing in and out of the interface. Many other tools also support SNMP monitoring. Simple Network Management Protocol (SNMP) covers a class of tools such as Multi Router Traffic Grapher (MRTG) and Cricket, which collect counter statistics from network infrastructures, and visualizes these statistics by means of graphs. This includes monitoring performed by Remote Monitoring (RMON) agents, which can be useful in determining the top hosts with regards to traffic, as well as the distribution of packet sizes on a network. SNMP can only be used to monitor devices that are SNMP managed, the reading and writing of the In Octet and Out Octet counters in a router could generate substantial traffic. Tcp dump prints out the headers of packets on a network interface that match the boolean expression. It is a network sniffer with in-built filtering capabilities; it can only collect the data from the network, but does not analyze collected data. The collected data can be analyzed offline with another utility namely, tcp show and tcp trace. As useful and powerful as tcp dump is, it is only suitable for troubleshooting i.e., for tracking network and protocol related connectivity problems [5][6].

MRTG is a versatile tool for graphing network data, this tool can run on a Web server. Every five minutes, it reads the inbound and outbound octet counter of the gateway router, and then logs the data to generate graphs for web pages. These graphs can be viewed using a web browser. Although MRTG gives a graphical overview, it however does not give details

about the host and protocol responsible for the traffic monitored. Windmill is a modular system for monitoring network protocol events; it is useful for acquiring the data from the network, but it is however limited in its capability by not providing any facility to aid in the analysis of those events or non protocol events acquired. WebTrafMon uses a probe to extract data from network packets and composes log files. Analysis results are based on the collected log files. Furthermore the user is able to view the analysis result via a generic web browser. WebTrafMon can show traffic information according to the source and destination host through any web interface; it can also show the traffic status according to each protocol in use. The continuous Query Systems share many of the concerns of other systems in acquiring and filtering continuous streams of data, this system however lack the ability to easily add new functions over the data, hence they are not extensible[7][8].

The agile and scalable analysis of network events is a system based on modular analysis and continuous queries allowing users to assemble modules into an efficient system for analyzing multiple types of streaming data. The system is optimized for analyzing and filtering large streams of data, and makes extensive use of polymorphic components that can perform common functions on new and unforeseen types of data without requiring any additional programming. This data analysis system provides a scalable, flexible system for composing ad-hoc analyses of high speed streaming data but does not provide infrastructures for data gathering and network monitoring[9].

Provides a display similar to the UNIX top command, but for network traffic, it can be used for traffic measurement and monitoring. Features such as the embedded HTTP server, support for various network media types, light CPU utilization, portability across various platforms, and storage of traffic information into an SQL database makes ntop versatile. The extensive cache usage has the drawback that memory usage is increased. This makes ntop a memory and computational intensive application. Like several of the other tools, ntop uses the same packet capture library to obtain the network data. The system is limited by not providing a universal web interface to display the results of its monitoring. The system requires huge storage, 11TB of shared disk space for data repository since it is not web based and also there is high overhead since the storage system is accessed through an NFS server over an Ethernet link [10].

To understand network dynamics, and to analyze a wide range of network behaviors, network data needs to be collected over a long period of time, rather than as a onetime event to capture transient behaviors that provide insight into immediate network problems. Network monitoring and analysis have become important topics in networking environments, encouraging worldwide research and development. Many organizations have developed automated network monitoring and analysis systems, some of which are described below [11][12].

The SunOS operating system provides Etherfind, which is a software packet monitor. The software opens the network card in the promiscuous mode and writes a summary line of each packet to a file. Information includes protocol type, size, and sending and receiving addresses. This tool extracts information from each packet. Data is supplied as a text-based user interface, and only users with root permission can access the tool. NFSwatch monitors all incoming network traffic destined to NFS file servers, and divides it into several categories. The number and percentage of packets received in each category is displayed on the screen, which is continuously updated. By default, NFSwatch monitors all packets destined for the current host. An alternate destination host to watch for may be specified using an internal argument. If a source host is specified with the -src argument, then only packets arriving at the destination host which were sent by the source host are monitored. This tool, like Etherfind, is inappropriate for our needs because it was originally designed to monitor a single host [13][14].

Etherload is a freely available LAN traffic analyzer for MS-DOS system with an Ethernet or Token Ring controller. This program was originally developed for measuring traffic load of Ethernet segments. Etherload basically captures each packet running through the LAN and provides various information on the packet. The program displays important parameters, events and loads for the protocols as well as the simple overall load statistics. Etherload can be used to check which host is generating the most traffic, which host is sending to which host, and what kind of protocols are in use in a specific Ethernet segment. It provides detailed parameter information and statistical data of the segment traffic. Especially for the TCP/IP network, Etherload is able to analyze and classify traffic by TCP/UDP port numbers, which give statistical information for various TCP/IP applications. After a careful examination of the above-mentioned tools, it was found that none of them provided the capability required for accurate network monitoring and analysis. This motivated the development of our new monitoring tool [15].

B. Packet Sniffing

A packet sniffer is also known as a network sniffer, network analyzer or protocol analyzer, or for particular types of networks, an Ethernet sniffer or a wireless sniffer. It is computer software or a computer hardware that can intercept and log traffic passing over a digital network or part of a network. As data flows across the network and the sniffer captures each packet and eventually decodes and analyzes its content. The network adapter must be put into promiscuous mode in order to capture the traffic which can be unicast, multicast or, broadcast type, and is being sent to the machine on which the sniffer software is running. Promiscuous mode or promiscuous mode is configuration of a network card that makes the network interface card pass all traffic it receives to the central processing unit rather than just packets addressed to it [16].

As promiscuous mode can be used to sniff on a network, one might be interested in detecting network devices that are in promiscuous mode. If a network device is in promiscuous mode, the kernel will receive all network traffic, i.e., the CPU load will increase. Then the latency of network responses will also increase, which can be detected. Promiscuous mode is often used to diagnose network connectivity issues. There are programs that make use of these features to show the user all the data being transferred over the network. The versatility of packet sniffer means they can be used to:

- Analyze network problems.
- Detect network intrusion attempts.
- Gain information for effecting a network intrusion.
- Monitor network usage.
- Gather and report network statistics.
- Debug client/server communications.

- Debug network protocol implementations.

C. *Protocols*

Hypertext Transfer Protocol is used for retrieving inter-linked resources led to the establishment of the World Wide Web. It is a application layer protocol for distributed, collaborative, hypermedia information system. HTTP is a request/response standard between a client and a server. The client making a HTTP request is referred to as user agent. The responding server which stores or creates resources such as HTML files and images is referred to as origin server. In between the user agent and the origin server there may be several intermediaries, such as proxies, gateway and tunnel. Typically, an HTTP client initiates a request .It establishes a TCP connection to a particular port on a host. An HTTP server listening on that port waits for the client to send a request message. Upon receiving the request , the server sends back a status line and a message of its own, the body of which is perhaps the requested resource, an error message or some other information[17][18].

The Transmission Control Protocol is one of the core protocols of the Internet Protocol Suite. TCP is so central that the entire suite is often referred as "TCP/IP". Whereas IP handles lower-level transmissions from the computer to computer as a message makes its way across the Internet, TCP operates at a higher level , concerned only with the two end systems, for example a Web browser and a Web server. TCP provides reliable, ordered delivery of a stream of bytes from one program on one computer to another program on another computer. Besides the web other common application of TCP includes e-mail and file transfer. Among its management tasks, TCP controls message size, the rate at which message are exchanged and network traffic congestion. TCP provides a communication service at an intermediate level between an application program and the Internet Protocol [19].

Due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets can get lost or delivered out of order. TCP detects these problems, request transmission of lost packets, rearranges out-of-order packets, and even helps minimize network congestion o reduce the occurrence of the other problems. Once the TCP receiver has finally reassembled a perfect copy of the data originally transmitted, it passes that datagram to the application program.

User Datagram Protocol is one of the core members of the Internet Protocol Suite, the set of network protocols used for the internet. With UDP, computer applications can send messages, in this case referred to as datagram's, to other hosts on an IP network without requiring prior communications to set up special transmission channels or data paths.UDP is sometimes called the Universal Datagram Protocol. TCP provides connections that need to be established before sending data. TCP connections have three phases:

1. Connection establishment
2. Data transfer
3. Connection termination

This is the reason why data transfer through UDP is faster compared to TCP protocol but in TCP data transmit is much more reliable. UDP provides an unreliable service and datagram's may arrive out of order, appear duplicated, or go missing without notice.UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead at the network interface level. UDP's stateless nature is also useful for servers that answer small queries from huge number of clients. Unlike TCP, UDP is compatible with packet broadcast and multicasting. Common network applications that use UDP include: Domain Name server (DNS), streaming media applications.

The IP is a protocol used for communicating data across a packet-switched internetwork using the Internet Protocol suite. IP is the primary protocol in the Internet layer and has the task of delivering distinguished protocol datagram's (packets) from the source host to the destination host solely based on their addresses. For this purpose the internet protocol defines addressing methods and structures for datagram encapsulation. The first major version of addressing protocol, now referred to as Internet Protocol Version 4(IPV4) is still dominant protocol of the internet. Data from an upper layer is encapsulated as packets. Circuit setup is not needed before a host may send packet to another host that it has previously not communicated with. Thus IP is a connection less protocol. Routers in the transmission path simply forward packets to next known local gateway matching the routing prefix for destination address [20][21].

Internet Protocol provides best effort delivery and its service can also be characterized as unreliable. In the network architectural language it is a connection less protocol, in contrast to so called connection oriented modes of transmission. The lack of reliability allows any of the following fault events to occur like data corruption, loss of data packet, duplicate arrivals, out of order delivery, meaning if a packet 'A' is sent before packet 'B', 'B' may arrive before packet 'A'. Since routing is dynamic and there is no memory in the network about the path of the prior packet, it is possible that the first packet sent takes a longer path to its destination [22].

The only assistance that the Internet protocol provides in Version 4(IPV4) is to ensure that IP packet header is error free through computation of checksums at routing nodes. This has the side effect of discarding packets with bad headers on the spot. In this case no notification is required to be sent to either node, although a facility exists in the Internet Control Message Protocol (ICMP) to do so. Perhaps the most complex aspects of IP are IP addressing and routing. Addressing refers to how end hosts become assigned to IP addresses and how sub networks of IP host addresses are divided and grouped together. IP routing is performed by all hosts, but most importantly by internetwork routers, which typically use either interior gateway protocols (IGPs) or external gateway protocols (EGPs) to help make IP datagram forwarding decisions across IP connected networks.

D. *Layers*

The Open Systems Interconnection model (OSI model) is a product of the Open Systems Interconnection effort at the International Organization for Standardization. It is a way of sub-dividing a communications system into smaller parts called layers. A layer is a collection of conceptually similar functions that provide services to the layer above it and receives services from the layer below it. On each layer an instance provides services to the instances at the layer above and requests service from the layer below. For example, a layer that provides error-free communications across a network provides the path needed by applications above it, while it calls the next lower layer to send and receive packets that make up the contents of

the path. Conceptually two instances at one layer are connected by a horizontal protocol connection on that layer. For the system the data link layer and the network layer needs to be explored [23][24].

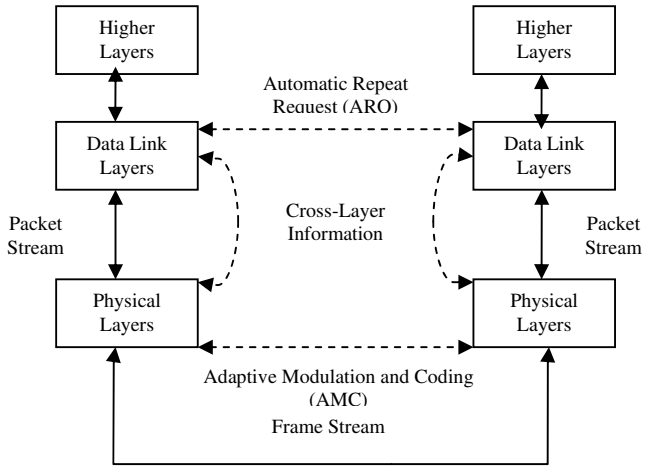

Figure 1: Communication Layer Structure

The Data Link Layer is concerned with local delivery of frames between devices on the same LAN. Data Link frames, as these protocol data units are called, do not cross the boundaries of a local network. Inter-network routing and global addressing are higher layer functions, allowing Data Link protocols to focus on local delivery, addressing, and media arbitration. In this way, the Data Link layer is analogous to a neighborhood traffic cop; it endeavors to arbitrate between parties contending for access to a medium.

When devices attempt to use a medium simultaneously, frame collisions occur. Data Link protocols specify how devices detect and recover from such collisions, and may provide mechanisms to reduce or prevent them. A frame's header contains source and destination addresses that indicate which device originated the frame and which device is expected to receive and process it. In contrast to the hierarchical and routable addresses of the network layer, layer 2 addresses are flat, meaning that no part of the address can be used to identify the logical or physical group to which the address belongs [25].

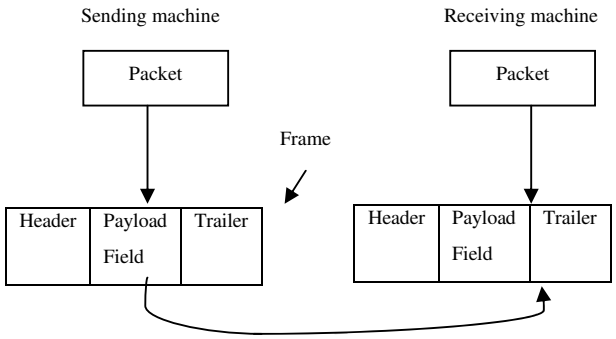

Figure 2: Mechanism of Packet Transfer

The data link thus provides data transfer across the physical link. That transfer can be reliable or unreliable; many data link protocols do not have acknowledgments of successful frame reception and acceptance, and some data link protocols might not even have any form of checksum to check for transmission errors. In those cases, higher-level protocols must provide flow control, error checking, and acknowledgments and retransmission.

The Network Layer is responsible for routing packets delivery including routing through intermediate routers, whereas the Data Link Layer is responsible for Media Access Control, Flow Control and Error Checking. The Network Layer provides the functional and procedural means of transferring variable length data sequences from a source to a destination host via one or more networks while maintaining the quality of service functions. Functions of the Network Layer include:

- Connection model: connectionless communication
  For example, IP is connectionless, in that a frame can travel from a sender to a recipient without the recipient having to send an acknowledgement. Connection-oriented protocols exist higher at other layers of that model.

- Host addressing
  Every host in the network needs to have a unique address which determines where it is. This address will normally be assigned from a hierarchical system, so you can be "Fred Murphy" to people in your house, "Fred Murphy, Main Street 1" to Dubliners, or "Fred Murphy, Main Street 1, Dublin" to people in Ireland, or "Fred Murphy, Main Street 1, Dublin, Ireland" to people anywhere in the world. On the Internet, addresses are known as Internet Protocol (IP) addresses.

- Message forwarding
  Since many networks are partitioned into subnetworks and connect to other networks for wide-area communications, networks use specialized hosts, called gateways or routers to forward packets between networks. This is also of interest to mobile applications, where a user may move from one location to another, and it must be arranged that his messages follow him. Version 4 of the Internet Protocol (IPv4) was not designed with this feature in mind, although mobility extensions exist. IPv6 has a better designed solution. Within the service layering semantics of the OSI network architecture the Network Layer respond to service requests from the Transport Layer and issues service requests to the Data Link Layer [26][27].

E. Packet Structure

Everything we do on the internet involves packet. For example, every web page we receive comes as a series of packets, and every e-mail we send leaves as a series of packets. Networks that ship data around a small packet are called packet switched networks. On the internet, the network breaks an e-mail message into parts of a certain size in bytes. These are the packets. Each packet carries the information that will help get to its destination – the senders IP address, the intended receivers IP address, something that tells the network how many packets this e-mail message has been broken into and the number of this particular packet. The packets carry the data in the protocols that the internet uses: Transmission Control Protocol/Internet Protocol (TCP/IP). Each packet contains part

of the body of your message. A typical packet contains perhaps 1,000 or 1,500 bytes.

Each packet is then sent off to its destination by the best available route – a route that might be taken by all the other packets in the message or by none of the other packets in the message. This makes the network more efficient. First, the network can balance the load across various pieces of equipment on a millisecond-by-millisecond basis. Second, if there is a problem with one piece of equipment in the network while a message is being transferred, packets can be routed around the problem, ensuring the delivery of the entire message. Depending on the type of network, packets may be referred to by another name: Frame, Block, Cell, and Segment. Most of the packets are further split into three parts:

Header- The header contains instructions about the data carried by the packet. These instructions may include:

- Length of packet
- Synchronization
- Packet Number
- Protocol
- Destination address
- Source address

Payload- Also called the body or data of a packet. This is actual data that the packet is delivering to the destination. If a packet is of fixed length, then the payload may be padded with blank information to make it the right size.

Trailer- The trailer, sometimes called the footer, typically contains a couple of bits that tell the receiving device that it has reached the end of the packet. It may also have some type of error checking. The most common error checking used in packets is Cyclic Redundancy Check (CRC). CRC is pretty neat. It takes the sum of all the 1s in the payload and adds them together. The result is stored in hexadecimal value in the trailer. The receiving device adds up the 1s in the payload and compares the result to the value stored in the trailer. If the values match, the packet is good. But if the values do not match, the receiving device sends a request to the source device to resend the packet. The 16-bit checksum field in the packet structure is used for error checking of header and data.

As an example, let's look at how an email message might get broken in packets. Let's say that the email is about 3.500 bits in size. The network we send it over uses fixed length packets of 1,024 bits. The header of each packet is 96 bit long and the trailer is 32 bit long, leaving 896 bits for the payload. To break the 3,500 bits of message into packets, we will need four packets. Three packets will contain 896 bits of payload and the fourth will have 812 bits. Each packet header will contain the proper protocols, the source and the destination address and the packet number. Routers in the network will look at the destination address and compare it to the lookup table to find out where to send the packet. Once the packet arrives at its destination, the destination computer will strip the header and the trailer off each packet and reassemble the e-mail based on the numbered sequence of the packets [28].

F. Technology Used

We have used Java technology to implement the system because the GUI that can be created from java is much simpler in approach and better than other languages. We have used the open source library of Jpcap and Winpcap to create the required system. Jpcap is java library used for capturing and sending network packets, also for rael time network traffic capture and analysis. Jpcap is based on LibPcap/WinPcap, and is implemented in C and JAVA. Jpcap has been tested on Microsoft Windows (98/2000/XP/Vista), and Linux. It is an API for developing packet capture applications in JAVA.

To understand how we capture packets we must know that Ethernet networks mostly use common bus topology, using either coax cable or twisted pair wire and hub. All of the nodes on the network can communicate over the same wire and take turns sending data using CSMA/CD. All the nodes on the network have their unique MAC address that they use to send packets of information to each other. If the network card is put into promiscuous mode it will look for all packets on the wire it is hooked to.

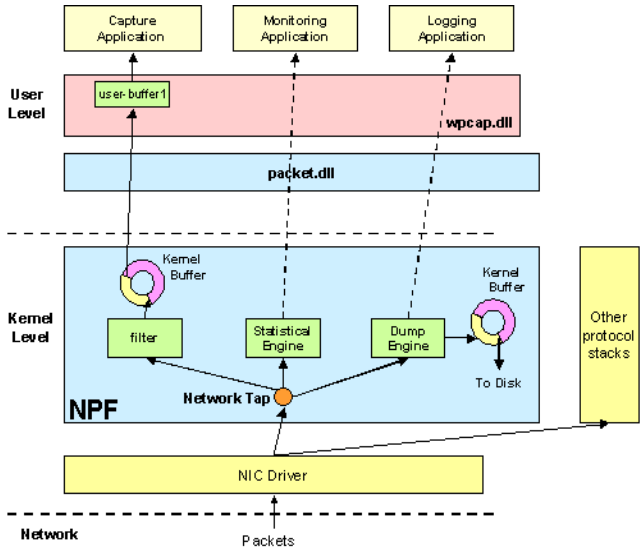

Figure 3: WinPcap Architecture to capture packets

The basic structure of WinPcap has the modules of a filtering machine, two buffers (kernel and user) and a couple of libraries at user level. The filtering process is started by a user-level component that is able to accept a user defined filter, compile them into a set of pseudo instructions, send these instructions to the filtering machine, and activate that code. On the other hand, the kernel module must be able to execute these instructions; there it must have a "BPF virtual machine" that executes the pseudo code on all the incoming packets. WinPcap is made up of three modules, one at kernel level and two at use level; user level modules come under the form of Dynamic link Libraries (DLLs) [29][30].

First module is the kernel part (NPF) that filters the packets, delivers them untouched to user level and includes some OS specific code.

Second module, packet.dll, is created to provide a common interface to the packet driver among the Win32 platforms. In fact, each Windows version offers different interfaces between kernel modules and user level applications: packet.dll deals with these differences, offering a system independent API. Programs based on packet.dll are able to capture packets on every Win32 platforms without being recompiled. Packet.dll includes several additional functionalities. It performs a set of low level operations like obtaining the adapters name or dynamic loading of the driver, and it makes available some system specific like the net mask of the machine and some hardware counters [31].

Third module, wpcap.dll, is not OS dependent and it contains some other high level functions such as filter generation and user level buffering, plus advanced features such as statistics and packet injection. Therefore programmers can have access to two type of API: a set of raw functions contained in packet.dll, which are directly mapped to kernel level calls, and a set of higher level functions that are provided by WPcap.dll and that are more user friendly and powerful [32].

III. MONITORING ARCHITECTURE

A. *Computational Model*

The client–server model of computing is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server machine is a host that is running one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.

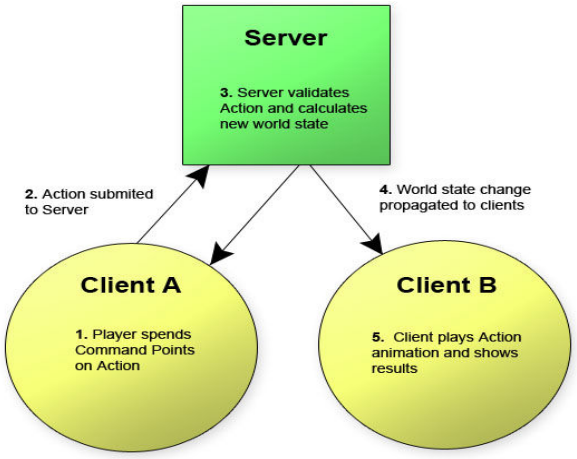

Figure 4: Client-Server Architecture for communication

The client–server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. Functions such as email exchange, web access and database access, are built on the client–server model. Users accessing banking services from their computer use a web browser client to send a request to a web server at a bank. That program may in turn forward the request to its own database client program that sends a request to a database server at another bank computer to retrieve the account information. The balance is returned to the bank database client, which in turn serves it back to the web browser client displaying the results to the user. The client–server model has become one of the central ideas of network computing. Many business applications being written today use the client–server model. So do the Internet's main application protocols, such as HTTP, SMTP, Telnet, and DNS. Specific types of clients include web browsers, email clients, and online chat clients[33][34][35].

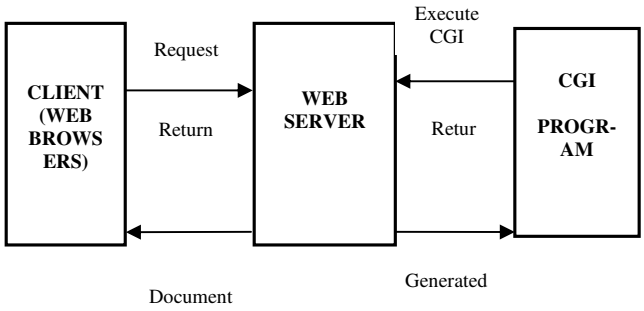

Figure 5: Architecture of Common Gateway Interface

The Web is based on client/server architecture and basically operates as follows. Hypertext documents that are to be made available on the Web are prepared in HTML and made accessible by the Web server. User wishing to retrieve the documents can do so by using a Web client (Web browsers such as Netscape Navigator or Internet Explorer) to connect to the Web server which contains the documents. The Common Gateway Interface (CGI) is the most popular method for interfacing external applications with Web servers [36].

With the potential for thousands of resources at geographically distant sites and tens-of thousands of simultaneous users, it is critical that data collection and distribution mechanisms scale. A general-purpose information management system such as an off-the-shelf database or directory service cannot efficiently meet this requirement because the characteristics of performance information are fundamentally different from the characteristics of the data these types of systems were designed to handle. In general, the following characteristics distinguish performance-monitoring information from other forms of system or program-produced data [37].

Performance information has a fixed, often short, lifetime of utility. Most monitoring data goes stale quickly, making rapid read access important but obviating the need for long-term storage. The notable exception to this is data that gets archived for accounting or postmortem analysis. Updates are frequent. Unlike the more static forms of "metadata," dynamic performance information is typically updated more frequently than it is read. Since most existent information-base technologies are optimized for query and not for update, they are potentially unsuitable for dynamic information storage. Performance information is often stochastic. It is frequently impossible to characterize the performance of a resource or an application component by using a single value. Therefore, dynamic performance information may carry quality-of-information metrics quantifying its accuracy, distribution, lifetime, and so forth, which may need to be calculated from the raw data. Systems that collect and distribute performance information should satisfy certain requirements [38][39][40]:

- Low latency: As previously stated, performance data is typically relevant for only a short period of time. Therefore, it must be transmitted from where it is measured to where it is needed with low latency.
- High data rate: Performance data can be generated at high data rates. The performance monitoring system should be able to handle such operating conditions.
- Minimal measurement overhead: If measurements are taken often, the measurement itself must not be

intrusive. Further, there must be a way for monitoring facilities to limit their intrusiveness to an acceptable fraction of the available resources. If no mechanism for managing performance monitors is provided, performance measurements may simply measure the load introduced by other performance monitors.

- Secure: Typical user actions will include queries to the directory service concerning event data availability, subscriptions for event data, and requests to instantiate new event monitors or to adjust collection parameters on existing monitors. The data gathered by the system may itself have access restrictions placed upon it by the owners of the monitors. The monitoring system must be able to ensure its own integrity and to preserve the access control policies imposed by the ultimate owners of the data.
- Scalable: Because there are potentially thousands of resources, services, and applications to monitor and thousands of potential entities that would like to receive this information, it is important that a performance monitoring system provide scalable measurement, transmission of information, and security.

In order to meet these requirements, a monitoring system must have precise local control of the overhead and latency associated with gathering and delivering the data. We believe that data discovery needs to be separated from data transfer if this level of control is to be achieved [41][42].

IV. ALGORITHM IMPLEMENTATION

A. To implement the proposed architecture of computational model following steps of algorithm required:

Step 1: Obtaining the list of network interfaces

1.1 Create an array of objects of NetworkInterface class names 'devices'
1.2 Use 'JpcapCaptor.GetDevicesList' function to detect the network interface device list.
1.3 Store the above list in the 'devices' variable.

Step 2: Displaying the list of network interface

2.1 Put the integer variable i=0
2.2 Till the value of i is less than the length of array 'devices' i.e, till i < devices. length goto Step 2.3 else goto Step 3
/* Here, it is noted that length of array devices Signify the number of network interface (NI) detected */
2.3 Print out the name of the captured NI.
System.out.println (devices[i].name+" ("+ Device[i].description+"));
2.4 i=i+1 i.e incrementing the value of i
2.5 Goto step 2.2

Step 3: Open a network interface with Open Device Function

3.1 Put the integer variable J=0
3.2 Till J<device.length goto step 3.3 else goto step 3.8
3.3 int index = J ; // set index of the interface that we want to open.
3.4 JpcapCaptor captor ; // create an instance of Jpcaptor class.
3.5 captor = JpcapCaptor.openDevice (device[index], 65535, false, 20);
/* Here parameters passed are in the function are: JpcapCaptor.openDevice(NetworkInterface interface , int snaplen, Boolean promics, int to_ms). */
3.6 J=J+1
3.7 Goto 3.2
3.8 Step6

Step 4: Capture packets from the network interface

4.1 Is the menu button of stop capture packet selected. If yes goto Step 3.8 else Step 4.2
4.2 Capture a single packet
String str = captor.getpacket( );
// We are calling getpacket( ) method of JpcapCaptor class multiple times to capture consecutive packets.
4.3 print packet by going to Step 5

Step 5: Close all the open network interface

7.1 Close the file i.e, writer.close();
7.2 captor.close();

Step 6: End

B. Pseudocode for opening of network interface using JPCAP functions:

```
//Obtain the list of network interfaces
NetworkInterface[ ] devices =
                        JpcapCaptor.getDevicesList ( );
//For each network interface
For(i = 1 to device.length)
//print out its name and description
System.out.println(i+":"+devices[i].name+"("+devices[i].
                                            description+");
//print out its datalink name and description
System.out.println("datalink:"+devices[i].datalink_name
          +"("+devices[i].datalink_description+"));
//print out the MAC address
System.out.println("MAC address:");
For (byte b: device[i].mac_address)
System.out.print(Integer.toHexString(b&0xff)+":");
System.out.println();
/*print out its IP address, subnet mask and broadcast
Address */
For (NetworkInterfaceAddress a:device[i].addresses)
System.out.println("address:"+a.address+" "+ a.subnet
                                        +" "a.broadcast);
```

The following steps obtain the list of network interfaces and prints out their information:

1. First of all prepare an ArrayList of all the network interfaces on the node.
2. Now for getting the device description we have to print the datalink description, MAC address. IP address, Subnet mask in a file. And then print all the data of the file.
3. Finally we will take anyone interface open it and will check for the packets entering through it.

V. EXPERIMENTAL RESULTS

To measure the impact of the monitoring system on the target application. It shows how monitoring can help to identify performance hotspots in the application design. For testing purposes, a small web application has been built. Users use a web browser to load in the web application, which has a simple form. After completing the form, the user submits it and then it call the methods of the web application in the LAN.

The overhead of the monitoring system on the global application response time has been measured. The results are shown in figure 6. The dotted lines describes the response time when monitoring is enabled. This test shows that with a large increase in the number of simultaneous users, the monitoring process does not affect the application response time significantly. This is largely due to the efficient use of the messaging infrastructure that enables the monitoring component to induce minimum overhead when notifying the listeners.

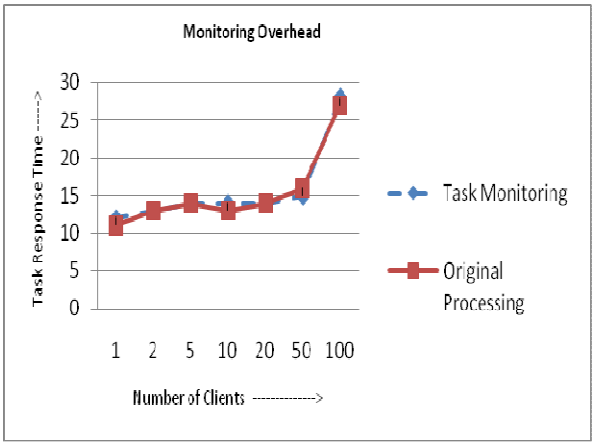

Figure 6: Performance of monitoring Overhead

We have analyzed that monitoring framework can be used to identify the causes of the increase in execution time presented in figure 5.

VI. CONCLUSION AND FUTURE WORK

The application system has been developed as one of the framework of web-based network traffic monitoring and analysis system. This application framework has been operated over a manageable network snooping and packet analysis system, and provides easy-to-use, omnipresent Web user interface. It provides network managers with real-time and historic traffic monitoring capabilities based on destination, source and source-destination host pair as well as based on network-layer, transport-layer and application-layer protocols.

The application will easily provide answers to questions such as "which application is consuming the most of network resources?", "and "which host pairs are generating the most network traffic?". This ability will provide more comprehensive insight into the network's use. Further, it can be extended this application system in such a way so that it can monitor and analyze switched networks (such as Fast Ethernet and Gigabit Ethernet).

VII. REFERENCES

[1] Jeffrey Joyce, Greg Lomow, Konrad Slind, Brian Unger, "Monitoring Distributed System", ACM Transactions on Computer System, ACM Trans. Comput. Syst., Vol. 5, No. 2. (May 1987), pp. 121-150.

[2] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson. A Monitoring Sensor Management System for Grid Environments. Proceedings of the IEEE High Performance Distributed Computing Conference (HPDC-9), August 2000.

[3] Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[4] CORBA. Systems Management: Event Management Service. X/Open Document Number:P437. http://www.opengroup.org/onlinepubs/008356299/.

[5] R. Housely, W. Ford, W. Polk, and D. Solo, "Internet X.509 Public Key Infrastructure," IETFRFC 2459. Jan. 1999

[6] R. Ribler, J. Vetter, H. Simitci, and D. Reed. Autopilot: Adaptive Control of Distributed Applications. Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, July 1998.

[7] W. Smith. A Framework for Control and Observation in Distributed Environments. NASA Advanced Supercomputing Division, NASA Ames Research Center, Moffett Field, CA.NAS-01-006, June 2001.

[8] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter. The Net Logger Methodology for High Performance Distributed Systems Performance Analysis . Proceedings of IEEE High Performance Distributed Computing Conference, July 1998.http://www-didc.lbl.gov/NetLogger/.

[9] Waheed, W. Smith, J. George, and J. Yan.An Infrastructure for Monitoring and Management in Computational Grids. In Proceedings of the 2000 Conference on Languages, Compilers, and Runtime Systems, 2000.

[10] M. Wahl, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3). Available from ftp://ftp.isi.edu/in-notes/rfc2251.txt.

[11] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Future Generation Computing Systems, 1999. http://nws.npaci.edu/.

[12] http://cseminar.web.cern.ch/cseminar/2002/2002-OtherFormats /t-20020716a.ppt

[13] http://datatag.web.cern.ch/datatag/WP3/grid_app_mon/net logger.htm

[14] http://acs.lbl.gov/NetLoggerWiki/index.php/NetLogger_Toolkit

[15] http://www-didc.lbl.gov/Talks/NetLogger.tutorial. HPDC99.pdf

[16] http://www-didc.lbl.gov/papers/chep.2K.Netlogger.pdf

[17] http://www.ace.ual.es/~jaberme/docsppal/cluster/DstrbtdPrllSystemsCluster_Grid.pdf

[18] http://www.escholarship.org/uc/item/3654d4bw?display=all#page-1

[19] http://www.computer.org/portal/web/csdl/abs/proceedings/hpdc/1998/8579/00/85790260abs.htm

[20] http://www.elsevier.com/wps/find/journaldescription.cws_home/505617/description#description

[21] http://www.gridcomputing.com/

[22] https://computing.llnl.gov/tutorials/parallel_comp/

[23] http://searchdatacenter.techtarget.com/sDefinition/0,sid80_gci773157,00.html

[24] http://rajith.2rlabs.com/2008/07/23/5-reasons-why-distributed-systems-are-hard-to-develop/

[25] http://portal.acm.org/citation.cfm?id=800056.802068

[26] http://lincolnventures.org/images/HPCA_paperRev1.pdf

[27] http://www.nec-labs.com/~gfj/saboori-icdcs-08.pdf

[28] http://www.globus.org/alliance/publications/papers/xuehaiMP04.pdf

[29] http://ganglia.info/papers/science.pdf

[30] http://arxiv.org/ftp/cs/papers/0410/0410012.pdf

[31] http://datatag.web.cern.ch/datatag/pfldnet2003/papers/tierney.pdf

[32] http://lcb99.in2p3.fr/BTierney.pdf

[33] http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf

[34] http://publib.boulder.ibm.com/infocenter/txformp/v6r0m0/index.jsp?topic=/com.ibm.cics.te.doc/erziaz0015.htm

[35] http://www.cs.helsinki.fi/u/jakangas/Teaching/DistSys/DistSys-08f-1.pdf

[36] http://cnx.org/content/m31661/latest/

[37] http://www.ida.liu.se/~TDDB37/lecture-notes/lect2-3.frm.pdf

[38] http://people.brunel.ac.uk/~csstrmh/research/distributed_testing.html

[39] http://www.globus.org/alliance/publications/papers/ogsa.pdf

[40] http://portal.acm.org/citation.cfm?id=1286589#

[41] http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01628415

[42] http://www.stickyminds.com/sitewide.asp?ObjectId=1462&Function=edetail&ObjectType=ART